**The first part is setting up access to Kaggle**

!mkdir ~/.kaggle

mkdir = make directory

**Creates a folder .kaggle (hidden) in your home directory (~= /root in Colab).**

**Put the file with your key (kaggle.json) in this folder.**

!mv kaggle.json ~/.kaggle/

mv = move.

**Moves the uploaded kaggle file.json to the created folder ~/.kaggle/.**

**After that, the Kaggle API knows where to look for the key (by default, it searches there)**

 **!**  
— означает: выполнить команду операционной системы (shell), а не Python-код.  
Всё, что идёт после !, выполняется как в терминале.

 **ls**  
— сокращение от list. Показывает список файлов и папок.

 **/content**  
— это **рабочая папка Google Colab**, где хранятся все файлы в вашей текущей сессии.  
Когда вы загружаете файлы, распаковываете архивы, скачиваете данные с Kaggle — всё оказывается именно там.

**1. numpy (np)**  
— библиотека для работы с **числовыми массивами и математикой**.  
Комментарий # linear algebra указывает, что её часто используют для линейной алгебры (векторы, матрицы, статистика).

**2. pandas (pd)**  
— библиотека для **работы с таблицами** (DataFrame).  
Позволяет читать, фильтровать, объединять и анализировать данные из CSV, Excel, SQL и т.д.

**3. sqlite3**  
— встроенный модуль Python для работы с **SQLite-базами данных**.  
Пока он импортирован «на будущее» — возможно, вы потом будете сохранять данные в базу или выполнять SQL-запросы.

**📂 Загрузка данных (CSV → DataFrame)**

Каждая из следующих строк:

df\_olist\_customers = pd.read\_csv('/content/olist\_customers\_dataset.csv')

читает CSV-файл с помощью pandas и сохраняет его содержимое в переменную DataFrame.

**Что это значит:**

* pd.read\_csv() — функция, которая **читает CSV-файл** (таблицу, разделённую запятыми).
* /content/... — путь к файлу в Google Colab (где вы его распаковали).
* df\_... — сокращение от *DataFrame*, принятое имя для таблиц в pandas.
* **📘 Что за файлы:**

| **Переменная** | **Файл** | **Что содержит** |
| --- | --- | --- |
| df\_olist\_customers | olist\_customers\_dataset.csv | информация о клиентах |
| df\_olist\_sellers | olist\_sellers\_dataset.csv | информация о продавцах |
| df\_olist\_order\_reviews | olist\_order\_reviews\_dataset.csv | отзывы покупателей |
| df\_olist\_order\_items | olist\_order\_items\_dataset.csv | позиции заказов (какие товары в каждом заказе) |
| df\_olist\_products | olist\_products\_dataset.csv | данные о товарах |
| df\_olist\_geolocation | olist\_geolocation\_dataset.csv | географические данные (координаты городов и регионов) |
| df\_product\_category\_name\_translation | product\_category\_name\_translation.csv | перевод названий категорий товаров |
| df\_olist\_orders | olist\_orders\_dataset.csv | информация о заказах (ID клиента, дата, статус) |
| df\_olist\_order\_payments | olist\_order\_payments\_dataset.csv | данные о платежах (тип оплаты, сумма и т.д.) |

This code is about **creating a temporary SQL database in memory**, then **storing pandas DataFrames as SQL tables** so you can query them using SQL.  
Let’s go through it line by line.

**🧩 1. Importing and creating a database engine**

from sqlalchemy import create\_engine

engine = create\_engine('sqlite://', echo=False)

**✅ What it does:**

* **SQLAlchemy** is a Python library that connects pandas to SQL databases (SQLite, PostgreSQL, MySQL, etc.).
* **create\_engine()** creates a **database engine** — a kind of “bridge” between Python and a SQL database.

The string 'sqlite://' means:

* Use **SQLite** (a lightweight, file-based SQL database).
* The empty path '' means “create the database **in memory**” (not saved on disk).  
  👉 it disappears when your Colab session ends.

The argument echo=False just tells SQLAlchemy **not to print every SQL command** to the console.  
If you set echo=True, you’ll see all SQL statements executed behind the scenes.

**🧱 2. Exporting pandas DataFrames to SQL tables**

Each line like this:

df\_olist\_customers.to\_sql("olist\_customers", con=engine)

means:

* **to\_sql()** → saves a pandas DataFrame as a table inside an SQL database.
* "olist\_customers" → is the **name of the table** that will be created.
* con=engine → tells pandas to use the connection (engine) you made above.

So after running all these lines, your **in-memory SQLite database** will have these tables:

| **Table name** | **Source DataFrame** |
| --- | --- |
| olist\_customers | df\_olist\_customers |
| olist\_sellers | df\_olist\_sellers |
| olist\_order\_reviews | df\_olist\_order\_reviews |
| olist\_order\_items | df\_olist\_order\_items |
| olist\_products\_dataset | df\_olist\_products |
| olist\_geolocation | df\_olist\_geolocation |
| product\_category\_name\_translation | df\_product\_category\_name\_translation |
| olist\_orders | df\_olist\_orders |
| olist\_order\_payments | df\_olist\_order\_payments |

**🧾 3. Viewing one of the DataFrames**

df\_olist\_order\_payments.head()

This last line does not query SQL — it just displays the first 5 rows of the original pandas DataFrame df\_olist\_order\_payments.

It’s a quick way to check that data is loaded correctly.

# 3. Query 1: Count and Percentage of Orders Purchased in Jan 2018 with 5 Review Score

1. **COUNT(o.order\_id)**
   * Считает количество заказов, удовлетворяющих основным условиям (o.order\_id — из olist\_orders, которые в JOIN с отзывами имеют review\_score = 5 и order\_purchase\_timestamp в январе 2018).
   * Результат — **целое число**.
2. **CAST(... AS FLOAT)**
   * Преобразует результат COUNT в **число с плавающей точкой**.
   * Важно, чтобы при делении не получалось целочисленное деление (иначе дробная часть теряется).
3. **SELECT COUNT(\*) FROM olist\_orders WHERE**

**strftime('%Y-%m', order\_purchase\_timestamp) = '2018-01'**

* + Это подзапрос, который считает **общее количество заказов в январе 2018**, без фильтра по отзывам.
  + strftime('%Y-%m', order\_purchase\_timestamp) извлекает **год и месяц** из даты заказа.

1. **Деление и умножение на 100:**

CAST(COUNT(...) AS FLOAT) / (SELECT COUNT(\*) ...) \* 100

* Считаем **долю заказов с 5-звёздочным отзывом** от общего числа заказов за январь 2018.
* Умножаем на 100, чтобы получить **процент**.

1. **ROUND(..., 2)**

* Округляет результат до **2 знаков после запятой**.
* Например, получаем 76.34 вместо 76.3432123.

1. **AS percentage**

* Задаёт имя столбца в результате SQL-запроса:

| percentage |

|------------|

| 76.34 |

**🔹 В итоге**

Эта конструкция:

1. Считает, сколько заказов в январе 2018 имеют **review\_score = 5**.
2. Делит на **общее число заказов за январь 2018**.
3. Преобразует в **процент** и округляет до двух знаков.

То есть получаем **процент 5-звёздочных заказов среди всех заказов в январе 2018**.